UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de un sitio web dinámico basado en base de datos sobre preferencias de juegos en niños y niñas de diferentes contextos sociales y culturales

Autor: Jorge Cámara Gómez
Tutor: Estrella Pulido

julio 2020

Jorge Cámara Gómez
Desarrollo de un sitio web dinámico basado en base de datos sobre preferencias de juegos en niños y niñas de diferentes contextos sociales y culturales

Jorge Cámara Gómez

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Por haber sido luz cuando todo estaba oscuro.
*Gracias, stack**overflow** [1].*

Misil más error igual a muerte.

D. V. Yakubovich

Resumen

La Fundación Educación y Desarrollo (FED) nació en 1994 con el objetivo de promover actividades que protegieran la educación y los derechos de las personas. Entre esos derechos se encuentra el derecho a jugar de los niños y las niñas en todo el mundo. La FED lleva años impulsando prácticas para estudiantes de diversas facultades en las que se recogen datos sobre preferencias de niños y niñas a la hora de jugar en colegios y ludotecas de diferentes contextos sociales y culturales.

En este trabajo se desarrolla una base de datos y una aplicación web con el objetivo de centralizar y modernizar la recogida y gestión de esos datos. De esta manera, se facilitará el análisis de los datos recogidos y la obtención de estadísticas interesantes que puedan promover estudios o investigaciones.

La aplicación dará acceso, en base al rol del usuario (administrador, investigador, profesor o estudiante), a distintas tablas con información sobre usuarios, prácticas, colegios, universidades, juegos y datos sobre preferencias en niños y niñas sobre juegos. Los profesores podrán dar de alta a sus estudiantes y ver su actividad, y los estudiantes podrán introducir los datos recogidos en sus prácticas.

En el proceso de desarrollo se han seguido las distintas fases de Análisis, Diseño, Implementación y Pruebas. La aplicación se ha desarrollado en Java, usando el framework Spring MVC y la herramienta Spring Boot. Para la base de datos se ha utilizado MySQL, y para hacer la interfaz de usuario más dinámica se ha usado el framework de JavaScript Vue.js.

Palabras clave

aplicación web, base de datos, juegos, Java, SQL, MySQL, Spring, Spring Boot, Maven, HTML, Bootstrap, JavaScript, Vue.js

Abstract

The Education and Development Foundation (EDF) was created in 1994 with the goal of promoting activities that protect education and human rights, among these rights being the right to play of all children in the world. The EDF has been offering internships for students from various areas of study for years, in which they collect data about the play preferences of children in schools and play centers in differing sociocultural contexts.

This project develops a database and web application with the objective of centralizing and modernizing the aforementioned data collection and management. By doing so, it facilitates the analysis of the collected data and reveals interesting statistics that could promote further studies or research.

The application will give access, depending on the user's role (administrator, researcher, professor, or student), to different tables of information about users, internships, schools, universities, games, and data about the play preferences of children. Professors can authorize their students and view their activity, and students can input the data collected in their practicum work.

The distinct phases of Analysis, Design, Implementation, and Testing were followed throughout the development process. The application was developed in Java, using the Spring MVC framework and the Spring Boot tool. The database was created using MySQL, and the JavaScript Vue.js framework was used to make the user interface more dynamic.

Keywords

web application, database, games, Java, SQL, MySQL, Spring, Spring Boot, Maven, HTML, Bootstrap, JavaScript, Vue.js

Índice

Listas

Lista de figuras

1

Introducción

Este Trabajo de Fin de Grado (TFG) se centra en el desarrollo de una **base de datos** y una **aplicación web** para la **Fundación Educación y Desarrollo (FED)** [2], una organización cuyo objetivo principal es promover el desarrollo de las personas a través del juego y la educación, cuyo logo podemos ver en la figura 1.1. Con esas herramientas se pretende modernizar y centralizar la recogida y gestión de los datos obtenidos por estudiantes de facultades como la de Educación o la de Psicología de la Universidad Autónoma de Madrid (UAM) que realizan prácticas en colegios y ludotecas sobre preferencias de niños y niñas a la hora de jugar y ocupar su tiempo libre. En este primer capítulo se hará una introducción del problema que se plantea y se explicarán los objetivos de este trabajo y la estructura de esta memoria.

Figura 1.1: Logo de la Fundación Educación y Desarrollo.

1.1. Motivación

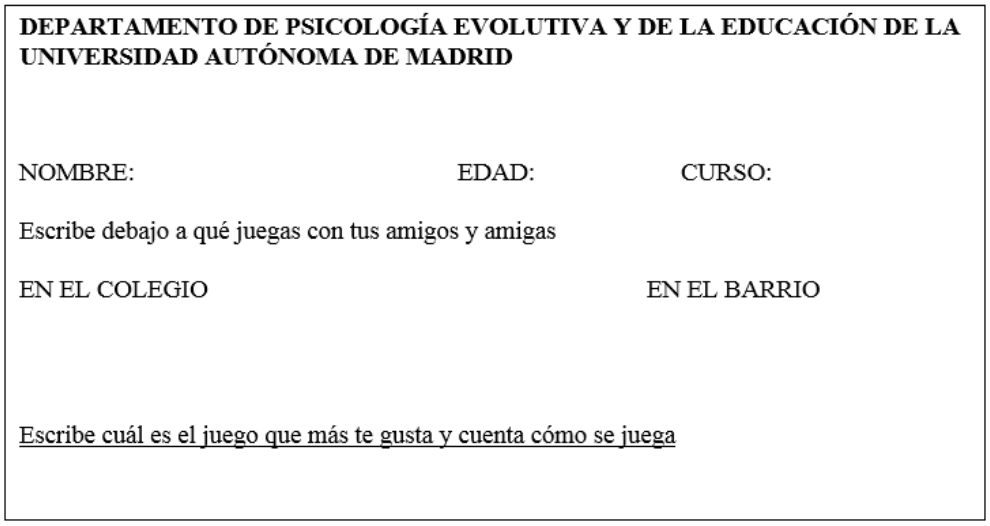
Desde que nacemos, los juegos forman parte esencial en nuestro crecimiento y desarrollo como personas. Desde actividades tan básicas como quitarse y ponerse un chupete, hasta jugar a los papás y las mamás o jugar a la pelota, con cada juego aprendemos algo. A través de ellos, los niños y niñas adquieren innumerables habilidades sociales, cognitivas, lingüísticas, motrices, etc.

Por ello, podemos considerar **jugar** no solo como una necesidad, sino también como un **derecho de la infancia** [3]. Si privamos a los más pequeños y las más pequeñas de la experiencia del juego, de relacionarse entre ellas en espacios seguros, les estaremos privando de todos esos aprendizajes y

experiencias que no podrían adquirir de otra manera y que también forman, junto con las instituciones regladas, parte de su educación.

Con esta premisa, nació en el año 1994 la **Fundación Educación y Desarrollo**. El objetivo principal de la fundación es promover la educación y los derechos de las personas, y colaborar con personas e instituciones para llevar a cabo actividades que coincidan en ese objetivo. Para ello, promueve la realización de prácticas, cursos, reuniones científicas o congresos relacionados con el desarrollo humano y la educación [2].

Dentro de todas estas actividades, una de las iniciativas que la FED ha impulsado a lo largo de su existencia es la recogida en colegios de varios paises de datos sobre **preferencias en niños y niñas a la hora de jugar**. Estos datos son recogidos por estudiantes en prácticas de titulaciones diversas como Educación Infantil y Primaria, Psicología, Antropología, etc. Podemos ver el formulario que los estudiantes pasan a los alumnos y alumnas de los colegios en la figura 1.2, y un ejemplo de los datos recogidos por un estudiante introducidos en un excel en la figura 1.3.



DEPARTAMENTO DE PSICOLOGÍA EVOLUTIVA Y DE LA EDUCACIÓN DE LA UNIVERSIDAD AUTÓNOMA DE MADRID

NOMBRE: EDAD: CURSO:

Escribe debajo a qué juegas con tus amigos y amigas

EN EL COLEGIO EN EL BARRIO

Escribe cuál es el juego que más te gusta y cuenta cómo se juega

Figura 1.2: Formulario original que responden los niños y las niñas en los colegios donde los estudiantes realizan prácticas. En él se recogen los juegos con los que juegan en el barrio y en el colegio, y cuál consideran que es su juego favorito, además del nombre, edad y curso del niño o niña.

La manera en la que hasta ahora se vienen recogiendo estos datos dificulta el análisis de los mismos. Cada estudiante en prácticas crea un excel distinto, y no se centralizan los datos recogidos de ninguna manera. El objetivo de este trabajo es crear una herramienta que permita a los estudiantes introducir la información recogida de forma sencilla para que todos los datos se almacenen en un mismo lugar, y poder así realizar análisis estadísticos o posibles investigaciones.

	A	B	C	D	E	F	G	H
1	**Identificación**	**código postal**	**nombre alumno**	**Sexo**	**Edad**	**Fútbol**	**Polis y cacos**	**Liebre**
2	7220901	28029	Sandra Güenes	1	11	**C**	**PC**	**B**
3	7220902	28029	Beatriz de la Torre	1	11	**B**	**C**	**PC**
4	7220903	28029	Patricia Pérez	1	11		**C**	**C**
5	7220904	28029	Natalia	1	11		**C**	**CB**
6	7220905	28029	Ángela Wolfe	1	11	**CB**		
7	7220906	28029	Isabel Marín	1	11	**B**	**C**	**C**
8	7220907	28029	Lourdes Bonilla	1	12	**CB**	**CB**	
9	7220908	28029	Marta	1	11	**C**		**PC**
10	7220909	28029	Naiara Rodríguez	1	11		**PC**	**CB**
11	7220910	28029	Carlos Soler	2	11	**PCB**	**C**	**B**
12	7220911	28029	Ricardo	2	11	**PCB**	**C**	**CB**
13	7220912	28029	José Javier Martín	2	11	**CB**	**C**	
14	7220913	28029	Adrián	2	11	**CB**	**C**	
15	7220914	28029	Alberto García	2	11	**CB**	**C**	
16	7220915	28029	Gonzalo Muratel	2	11	**PCB**	**CB**	**CB**
17	7220916	28029	Luis Méndez	2	11	**PCB**		
18	7220917	28029	Sergio Portillo	2	11	**CB**	**C**	**PB**

Figura 1.3: Extracto de uno de los excel donde se recogen los datos de los formularios. Los números de identificación son combinaciones únicas de IDs de colegios y niños. Las siglas P, C y B indican si el juego es el Preferido del niño o la niña, si lo juegan en el Colegio o en el Barrio.

1.2. Objetivos

El proyecto tiene como objetivo crear una base de datos que permita centralizar la información recogida en los formularios para que pueda ser analizada de manera global o con distintos filtros estadísticos que pudieran resultar interesantes: regiones, género, edades, etc. Para manipular esta base de datos, se creará una aplicación web a la que tendrán acceso usuarios con distintos roles como administradores, profesores o estudiantes.

Los estudiantes a los que profesores hayan asignado prácticas podrán usar la aplicación para introducir los datos recogidos en los formularios, pudiendo crear nuevas entradas para alumnos, juegos y registros de datos.

La idea inicial cuando se comenzó a realizar este trabajo era que los estudiantes pudieran probar una primera versión de la aplicación web durante el curso 2019/2020. Debido a la situación derivada de la pandemia por Covid-19, todas las actividades de prácticas se han suspendido, por lo que esta parte de pruebas y feedback más amplio no ha podido llevarse a cabo. Sin embargo, sí se han realizado pruebas a un nivel más reducido, que se explicarán en esta memoria.

1.3. Estructura

En esta memoria se explicarán las distintas fases por las que ha pasado el desarrollo de la aplicación, y se divide en siete capítulos:

1.– **Introducción**. En este primer capítulo hemos hecho una introducción de lo que será el

trabajo, viendo su contexto y objetivos principales.

2.– **Estado del arte**. Se describen posibles herramientas similares a la que se va a desarrollar en el trabajo.

3.– **Análisis**. Se realiza junto con miembros de la FED un trabajo de análisis y se establecen unos requisitos funcionales y no funcionales para la aplicación que se espera obtener.

4.– **Diseño**. Tras realizar el análisis de requisitos, se decide qué arquitectura va a seguir la aplicación, así como el modelo de datos y los aspectos fundamentales de la interfaz de usuario.

5.– **Desarrollo**. En este capítulo se analizan las tecnologías utilizadas para el desarrollo de la aplicación, y se detallan y explican las decisiones y procedimientos más importantes que se han llevado a cabo en esta fase del proyecto.

6.– **Pruebas y resultados**: Se exponen las pruebas que se han realizado durante y tras el desarrollo para asegurar el correcto funcionamiento de la aplicación y detectar posibles errores.

7.– **Conclusiones y trabajo futuro**: En el último capítulo se hace un resumen de lo que ha sido este trabajo, los objetivos que se han cumplido y se plantean propuestas de mejora y ampliación para una posible continuación del desarrollo de la aplicación.

2

Estado del arte

Jugar es algo que han hecho los niños y niñas a lo largo de toda la historia. Cuando combinamos el campo de los juegos y el de las tecnologías surgen innumerables posibilidades, y aparecen inevitablemente los videojuegos. Sin embargo, también se puede utilizar la tecnología para crear herramientas que nos ayuden a conocer más sobre los juegos, sobre las actividades con las que se desarrollan los niños y las niñas alrededor del mundo. En este capítulo vamos a describir algunas de esas herramientas.

2.1. APIs de videojuegos

Podríamos decir que vivimos en la era de los datos. Cada vez hay más recopilaciones de datos de todo tipo, bibliotecas digitales sobre cualquier tema. Y no es de extrañar que una de las áreas donde más está creciendo el volumen de datos y de consultas de estos datos sea en los juegos. Más concretamente, en los videojuegos. Porque es difícil actualmente buscar información sobre “juegos” sin que vaya implícito que se trata de videojuegos.

Existen por tanto numerosas herramientas para consultar características de videojuegos y todo lo relacionado con ellos. Muchas de estas herramientas son lo que se conoce como Application Programming Interface (API) , que en español se traduce como Interfaz de Programación de Aplicaciones. Estas interfaces ponen en conexión distintos sistemas, definiendo un conjunto de métodos o rutinas para acceder a ciertos servicios.

El mejor ejemplo de estas API es **IGDB** [4]. Se trata de una herramienta desarrollada por la plataforma de streaming de videojuegos Twitch [5] que da acceso a la mayor base de datos online de videojuegos, con la que se pueden consultar características, datos y estadísticas sobre más de 100.000 juegos. Al tratarse de una biblioteca abierta, cualquier persona o empresa puede utilizarla para obtener información, usarla para otros proyectos, e incluso completarla con nuevas aportaciones.

2.2. Bases de datos de juegos

Si salimos del mundo de los videojuegos, se reduce significativamente el número de organizaciones y herramientas dedicadas a los juegos no digitales. Sin embargo, los juegos físicos siguen siendo un pilar en la educación de los niños y las niñas, y afortunadamente hay alguna organización que trabaja para que este legado no quede en el olvido y siga habiendo información disponible sobre esas actividades.

Una de ellas es **Playworks** [6], una organización sin ánimo de lucro estadounidense fundada en 1995. Esta organización tiene objetivos muy similares a los de la Fundación Educación y Desarrollo, como potenciar y proteger los juegos como una necesidad y un derecho de los niños y las niñas, y crear entornos seguros donde puedan desarrollarse a través de estas actividades.

Además de tener una red de personas que promueven estos principios en más de 7.000 colegios alrededor de los Estados Unidos de América, han creado una **librería de juegos** muy interesante que puede consultar cualquier persona a través de su web [7]. En esta librería han reunido cientos de juegos de todos los tipos: juegos de pelota, de cooperación, de interior, de exterior, etc. La herramienta permite buscar juegos y filtrarlos tanto por nombre como por una serie de características como son el tipo de juego, el número de personas necesarias para jugarlo, las edades para las que se recomienda, los materiales necesarios o la duración del juego, como podemos ver en la figura 2.1.

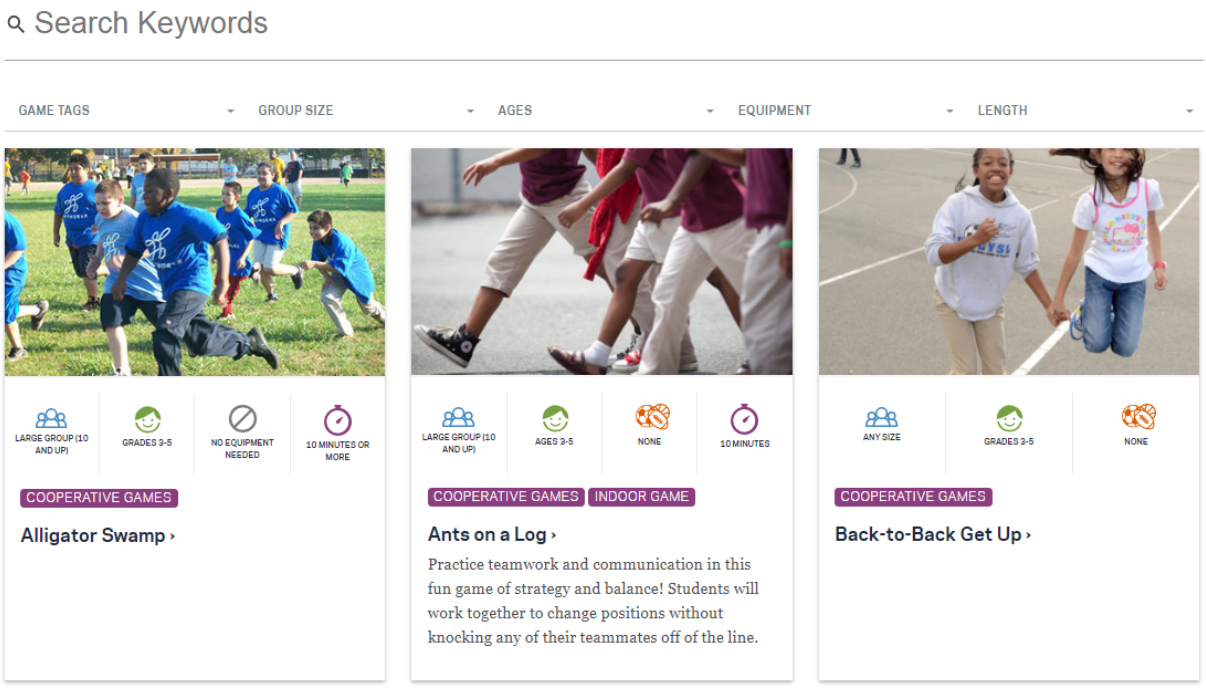


Figura 2.1: Herramienta de búsqueda de juegos de la librería de Playworks. Permite filtrar por nombre, tipo de juego, número de personas necesarias, edades, material necesario y duración.

2.3. Estudios sobre preferencias de niños y niñas en los juegos

Investigadores de áreas como la psicología o la educación han tenido claro desde hace mucho tiempo que los juegos constituyen una herramienta fundamental en el desarrollo cognitivo y social de los niños y las niñas. Por ello, podemos encontrar diversos estudios que analizan desde distintas perspectivas las preferencias de los más pequeños a la hora de jugar.

B. Sutton-Smith y B. G. Rosenberg [8] compararon en el año 1961 diversas investigaciones sobre preferencias de juegos en niños y niñas estadounidenses que se habían realizado durante los primeros sesenta años del siglo XX para analizar la importancia de los juegos en el folclore y ver la evolución de esas preferencias según el contexto histórico y los cambios que se iban produciendo. En el año 2008, Mable B. Kinzie y Dolly R. D. Joseph [9] hicieron hincapié en la perspectiva de género al analizar las preferencias de niños y niñas, y en el año 2018 Zeynep Tatli [10] realizó un estudio con 500 niños y niñas en el que se pone de manifiesto el aumento de la importancia de los videojuegos y las diferencias de género que siguen existiendo a la hora de jugar.

2.4. Conclusiones

Las herramientas existentes que ofrecen datos y estadísticas sobre juegos se centran principalmente en los videojuegos. Sobre ellos hay multitud de webs y bases de datos con APIs para acceder a toda la información que proporcionan. Sin embargo, cuando nos centramos en los juegos no digitales, la información disponible es mucho más escasa. Existen numerosos estudios acerca de las preferencias de los niños y las niñas en los juegos, pero no se ha encontrado ninguna herramienta abierta que permita recoger datos de manera colaborativa sobre este tema.

La aplicación web que se va a desarrollar en este trabajo cubre una necesidad muy específica: una herramienta que permita a estudiantes introducir los datos obtenidos en colegios y ludotecas sobre preferencias de niños y niñas a la hora de jugar. En principio, esta herramienta sería usada por los estudiantes que realizan prácticas con la Fundación Educación y Desarrollo, pero podría extenderse su uso de manera más general, llegando a reunir muchos datos de lugares muy diversos. A partir de estos datos se podrán realizar distintos análisis estadísticos.

3

Análisis

En este capítulo presentamos el análisis previo al diseño de la aplicación. Es una fase de vital importancia, ya que sienta las bases y los requisitos sobre los que se construirá el proyecto. Tan pronto como se asignó este TFG , se celebraron diversas reuniones con las personas responsables de la FED para empezar a delimitar lo que sería la aplicación, el público al que se dirigiría y las funcionalidades que tendría. De estas reuniones se extrae primero un diagrama de casos de uso, y posteriormente un análisis de requisitos más detallado.

3.1. Casos de uso

En la figura 3.1 podemos ver el Diagrama de Casos de Uso de la aplicación extraído de las primeras reuniones del proyecto. Este diagrama resume, de manera muy general, las funcionalidades que se pretenden obtener de la aplicación. Hay que dejar claro que la funcionalidad relativa al perfil de investigador es algo que se plantea en las reuniones, pero cuya realización no irá forzosamente incluida en la primera entrega del proyecto, que pretende más bien dejar sentadas las estructuras para poder añadir funcionalidades avanzadas más adelante. El objetivo principal de este trabajo, como se ha indicado, es crear una base de datos consistente y una aplicación base para trabajar sobre ella.

3.2. Maquetas

En el proceso de análisis también se ha hecho uso de otra herramienta que puede ser muy útil si el usuario no tiene del todo claros los requisitos del sistema o el alcance del proyecto. Estamos hablando de las maquetas, que permiten validar con el cliente o usuario algunos requisitos, siempre teniendo en cuenta que no tienen por qué ser iguales que el producto final. Tenemos un ejemplo de estas maquetas en la figura 3.2.

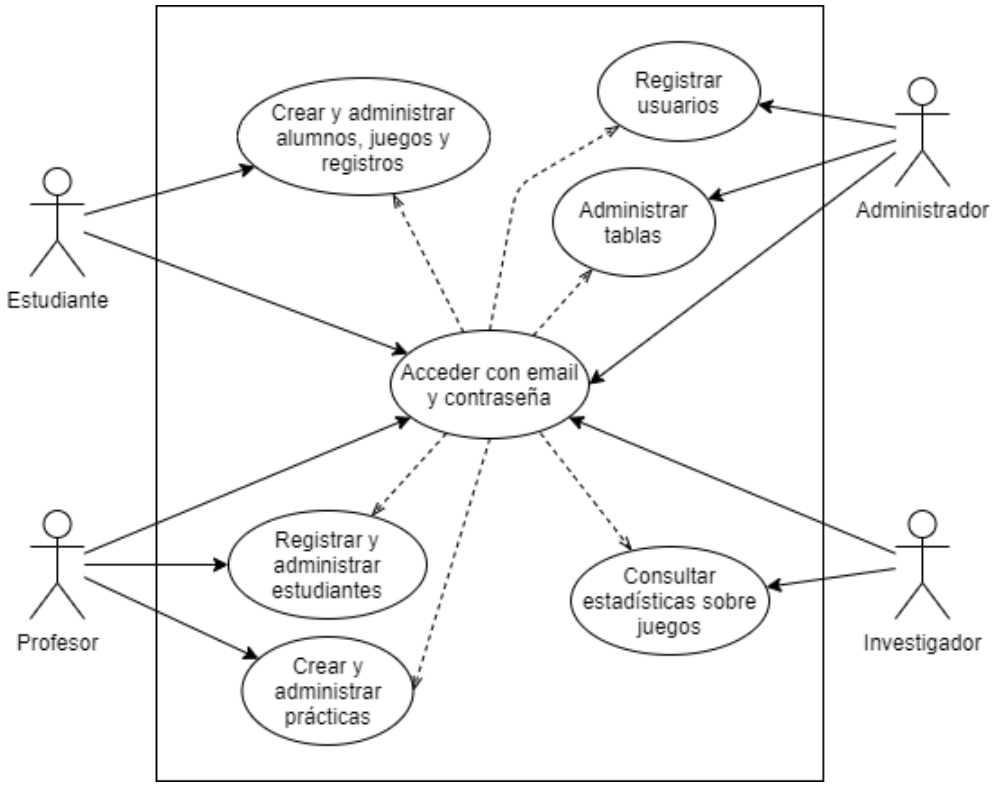


Figura 3.1: Diagrama de Casos de Uso de la aplicación, con los actores y acciones principales.

Figura 3.2: Maqueta del aspecto general de la aplicación, con el logo de la FED, vista de los menús principales y una tabla de ejemplo.

3.3. Análisis de requisitos

Una vez entendido el objetivo de la aplicación, se procede a formalizar las funcionalidades y otros aspectos de la misma en el siguiente análisis de requisitos. Vamos a dividir estos requisitos en funcionales y no funcionales. Los primeros describen de manera concreta las acciones fundamentales y esenciales para el correcto comportamiento del sistema; los segundos, características o cualidades generales que se esperan del software.

3.3.1. Requisitos funcionales

RF-1.– Los usuarios de la aplicación podrán entrar mediante un **sistema de login** para poder hacer uso de las funcionalidades que se ofrecen.

- **RF-1.1.–** El usuario introducirá email y contraseña, que se contrastarán con los usuarios almacenados en la base de datos.
- **RF-1.2.–** Si los credenciales son correctos, se accederá a la página principal de la aplicación. Si no, aparecerá un mensaje de error.
- **RF-1.3.–** En cualquier momento, el usuario podrá cerrar su sesión.

RF-2.– El sistema contará con distintos ***roles*** para los usuarios de la aplicación. Estos roles son ***Estudiante***, ***Profesor***, ***Investigador*** y ***Administrador***. Las características de cada rol se irán especificando en el resto de requisitos.

RF-3.– Todos los usuarios contarán, al menos, con los siguientes atributos: un *email*, una *constraseña*, un *rol* y una *universidad*.

RF-4.– La base de datos contendrá al menos un usuario con rol de *administrador*, y el **registro de nuevos usuarios** podrá hacerse con la siguiente jerarquía:

- **RF-4.1.–** Un *administrador* podrá dar de alta a nuevos *administradores*, *profesores* o *investigadores*, así como editar o eliminar cualquier usuario.
- **RF-4.2.–** Un *profesor* podrá dar de alta a *estudiantes*, y tendrá poder para editar o eliminar usuarios que él haya creado.
- **RF-4.3.–** *Estudiantes* e *investigadores* no podrán dar de alta nuevos usuarios ni tendrán acceso a la tabla de usuarios.

RF-5.– Los *profesores* podrán crear ***prácticas*** que podrán ser asignadas a los estudiantes.

- **RF-5.1.–** Cada práctica creada tendrá asignados, al menos, el colegio donde se va a realizar, el año de realización, el tipo de actividad sobre el que se va a hacer la práctica y la asignatura con la que se relaciona la práctica.
- **RF-5.2.–** En una práctica solo se recogerán datos de un ***tipo de actividad***. Se

proponen unos tipos de actividad iniciales, siempre con la posibilidad de añadir más: ***juegos***, ***videojuegos***, ***juguetes*** y ***deportes***.

RF-5.3.– La *práctica* será asignada al *estudiante* en el momento en que el *profesor* registre ese usuario en la aplicación.

RF-5.4.– El *profesor* puede dar por finalizada la *práctica* en cualquier momento, a partir del cual los estudiantes de esa *práctica* dejarán de tener acceso al sistema.

RF-6.– Las figuras de ***universidades*** y ***colegios***, que ya se mencionan en los requisitos RF-3 y RF-5.1, quedarán recogidas en el sistema.

RF-6.1.– Ambas tendrán, al menos, un nombre y una ***región*** asignados. Cada *región*, a su vez, tendrá asignado su ***país*** correspondiente, todo ello para permitir futuros estudios por regiones o países.

RF-6.2.– Los *administradores* podrán crear, editar y eliminar *universidades* y *colegios*. Los *profesores* podrán hacerlo solo con los *colegios*.

RF-7.– Los estudiantes que tengan asignada una *práctica* y que, por tanto, tengan acceso a la aplicación, podrán introducir los datos recogidos en el cuestionario modelo. Para ello, podrán:

RF-7.1.– Crear, editar y eliminar ***alumnos***. Los registros de datos serán sobre estos alumnos. Cada *alumno* tendrá, al menos, un curso, edad y género asignados.

RF-7.2.– Introducir en el sistema un nuevo ***registro de datos***. Para cada *juego* que el *alumno* mencione, el *estudiante* creará un nuevo *registro* que contendrá, al menos: el *alumno*, el *juego*, el *colegio*, una marca indicando si la realiza en el *barrio*, otra indicando si la realiza en el *colegio*, y otra indicando si es su actividad *favorita*.

RF-7.3.– Editar y eliminar los registros creados.

RF-7.4.– Crear nuevos ***juegos*** si no se encontraran ya en el sistema.

RF-8.– En todas las tablas, el borrado de datos se podrá hacer de manera individual o seleccionando varios elementos a la vez.

3.3.2. Requisitos no funcionales

RNF-1.– Requisitos de interfaz y usabilidad.

RNF-1.1.– La página principal de la aplicación para cada usuario consistirá en una lista con las tablas a las que tiene acceso. El usuario podrá ir seleccionando las distintas tablas para visualizar datos, y dentro de cada tabla podrá añadir, editar

o eliminar elementos si es que tuviera dichos permisos.

RNF-1.2.– Cuando en una tabla haya muchos elementos, se evitará tener que hacer scroll hacia abajo de manera excesiva en la web, bien con un sistema de paginación, bien con un scroll dentro de la propia tabla.

RNF-1.3.– Tras cualquier creación, edición o borrado de datos, aparecerá un mensaje informando del éxito o fracaso de la operación.

RNF-2.– Requisitos operacionales.

RNF-2.1.– El borrado de datos será un borrado lógico, para poder recuperar datos si fuera necesario. Se marcará temporalmente cuándo se realizó el borrado de cada elemento.

RNF-2.2.– Cada elemento presente en la base de datos tendrá una marca del usuario que lo creó y una marca temporal del momento de creación.

RNF-2.3.– Si un elemento presente en la base de datos es modificado, se marcará temporalmente esa modificación.

RNF-3.– Requisitos de seguridad.

RNF-3.1.– El acceso a la aplicación solo será posible con email y contraseña.

RNF-3.2.– Las contraseñas se codificarán en la base de datos.

RNF-3.3.– Como se ha ido mencionando en requisitos anteriores, cada *rol* tendrá acceso solo a ciertas tablas. Estos accesos se controlarán con un sistema de ***perfiles*** y **permisos** asignados a esos *perfiles*. Se evitará mostrar cualquier link de acceso a tablas para las que no se tiene permiso, y se impedirá también el acceso a través de urls.

RNF-4.– Requisitos de documentación.

RNF-4.1.– La aplicación se construirá de la manera necesaria para, más adelante, poder funcionar en diversos idiomas sin necesidad de grandes cambios en la arquitectura.

RNF-4.2.– Habrá algún tipo de *ayuda online* para los usuarios de la aplicación, donde se explique de forma sencilla cómo hacer uso correcto de ella.

RNF-5.– Requisitos de rendimiento.

RNF-5.1.– La aplicación soportará varios usuarios conectados a la vez realizando consultas sobre la base de datos.

RNF-6.– Requisitos de mantenibilidad.

RNF-6.1.– Se podrá modificar cualquier texto de la aplicación de manera sencilla sin tener que acceder al fichero html correspondiente.

4

Diseño

La segunda fase de un proyecto software es el Diseño. Esta fase es el proceso de definición de la arquitectura, componentes, datos de un sistema, etc, para satisfacer los requisitos especificados en la fase de Análisis. Si el Análisis trata el *qué*, el Diseño trata el *cómo*. En este capítulo vamos a ver ese proceso y sus resultados, que consisten en el diseño de una base de datos relacional y en las ideas de diseño principales para la interfaz de usuario de la aplicación.

4.1. Arquitectura

Al iniciar un proyecto hay que escoger una arquitectura software adecuada para el sistema que se pretende construir. Se trata de decidir las estructuras de más alto nivel, una abtracción que simplifique mucho el proceso de diseño y desarrollo. En este proyecto hemos optado por el patrón de arquitectura **Modelo-Vista-Controlador (MVC)** [11].

4.1.1. Modelo-Vista-Controlador

El patrón de arquitectura MVC distingue tres componentes en un sistema software:

- El **Modelo**: es la representación de los datos del sistema. Se encarga de gestionar todo tipo de acceso a la base de datos, sirviendo la información almacenada y encargándose de actualizarla cuando sea necesario siguiendo una lógica de negocio establecida según los requisitos del sistema.
- La **Vista**: es la parte del sistema visible para el usuario. Presenta los datos del modelo de manera adecuada y ordenada para que la interacción entre usuario y sistema sea cómoda.
- El **Controlador**: hace de nexo entre el modelo y la vista. Cuando un usuario realiza una petición a través de la vista, el controlador solicita al modelo la información necesaria y la sirve de nuevo a la vista.

Podemos ver un esquema del funcionamiento de este modelo con las relaciones descritas entre

sus componentes en la figura 4.1.

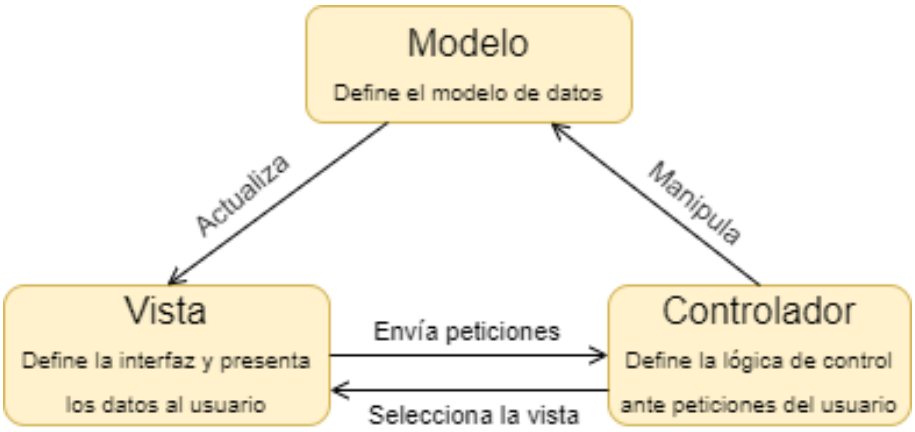


Figura 4.1: Diagrama de interacciones entre los componentes del patrón de arquitectura software Modelo-Vista-Controlador.

4.2. Modelo de datos

En esta sección vamos a explicar el diseño del modelo de datos de la aplicación. Necesitamos entidades que representen los distintos elementos de los que se hace uso. Tras realizar el análisis de requisitos, se identifican esas entidades, que se exponen y explican a continuación junto con sus atributos.

- **Usuario**: representa cada una de las personas que usarán la aplicación. Sus credenciales serán *email* y *password*, y podrá acceder a la aplicación si *activo* vale 1. Además, contiene una *Universidad*, un *Perfil* para controlar los permisos y, en caso de que sea un estudiante, tendrá una *Práctica* asignada.
- **Perfil**: representa los distintos roles que un usuario puede tener (RF-2), y que sirven para otorgar o denegar permisos en la aplicación.
- **Menu**: entidad que representa cada una de las tablas de la aplicación. Contiene el *nombre* de la tabla y la *url* de acceso en la aplicación.
- **Permiso**: esta entidad, que sirve de nexo entre *Perfil* y *Menu*, contiene los permisos que se conceden a cada tipo de *Perfil* para cada tabla. Si existe una tupla con un *Perfil* y un *Menu* concretos, querrá decir que los usuarios con ese perfil tienen acceso a esa tabla.
- **Práctica**: representa las prácticas que los profesores pueden crear para asignar a sus estudiantes (RF-5). Cada una contiene una *asignatura*, el *TipoActividad* sobre el que será la práctica, el *Colegio* donde se realizará, el *Año* en que se hará, y un *Usuario* que será el profesor responsable.
- **Tipo Actividad**: representa las distintas actividades sobre las que se puede hacer una

práctica, tal y como se especifica en el requisito RF-5.2.

- **Juego**: representa las actividades que podrían ser nombradas por los alumnos y alumnas.
- **Alumno**: representa los alumnos y las alumnas que responderán el formulario base y proporcionarán datos a los estudiantes en prácticas para introducir en la aplicación. Contienen un *nombre*, una *edad*, un *curso* y un *género*. Estos tres últimos atributos serán de vital importancia para analizar los datos recogidos.
- **Registros**: representación de la información aportada por un alumno o alumna sobre un juego. Cada *Registro* tiene el *Alumno*, *Colegio* y *Año* en que se obtuvieron los datos y el *Juego* nombrado. Además, contiene tres atributos que indican si el alumno o alumna realiza la actividad en el *barrio*, en el *colegio*, y si se trata de su actividad *favorita*. Estos tres valores no son excluyentes.
- **Colegio**: representa los colegios donde se realizarán las prácticas y se recogerán datos. Cada uno contiene su *nombre*, *direccion* y *localidad*, y la *Region* a la que pertenece.
- **Universidad**: representa cada una de las universidades que usen la aplicación. Contiene los mismos atributos que *Colegio*.
- **Curso**: cada uno de los cursos en los que se realiza la recogida de datos.
- **Año**: representa el año escolar en el que se realiza una *práctica* y se recogen *registros*.
- **Región**: representa las regiones donde pueden ubicarse los colegios y universidades. Cada una contiene el *Pais* al que pertenecen.
- **País**: representa los países donde se ubican las regiones anteriormente nombradas.

Además de todos los atributos explicados, cada entidad contendrá tres atributos de tipo *Timestamp* que registrarán el momento de creación, modificación o baja de cada instancia: *fechaAlta*, *fechaBaja* y *fechaModificacion*. También contendrán una referencia a un *Usuario*, que servirá de marca para saber quién es el autor de cada tupla. Estos atributos harán cumplir los requisitos indicados en FNF-2.

Por supuesto, cada entidad contendrá también un identificador, *id*, que será un entero auto generado y único, y que actuará de clave primaria de la entidad.

Podemos ver todas estas entidades y sus relaciones en el diagrama de clases de la figura 4.2.

4.3. Interfaz

La interfaz de la aplicación será visualmente sencilla e intuitiva. Al acceder encontraremos una pantalla de login con el logo de la FED y una breve explicación de qué es esta aplicación. En la página principal el usuario verá las distintas tablas a las que tiene acceso y podrá ir accediendo a ellas a través de un menú en la parte superior. En la parte izquierda de la pantalla veremos un menú en el

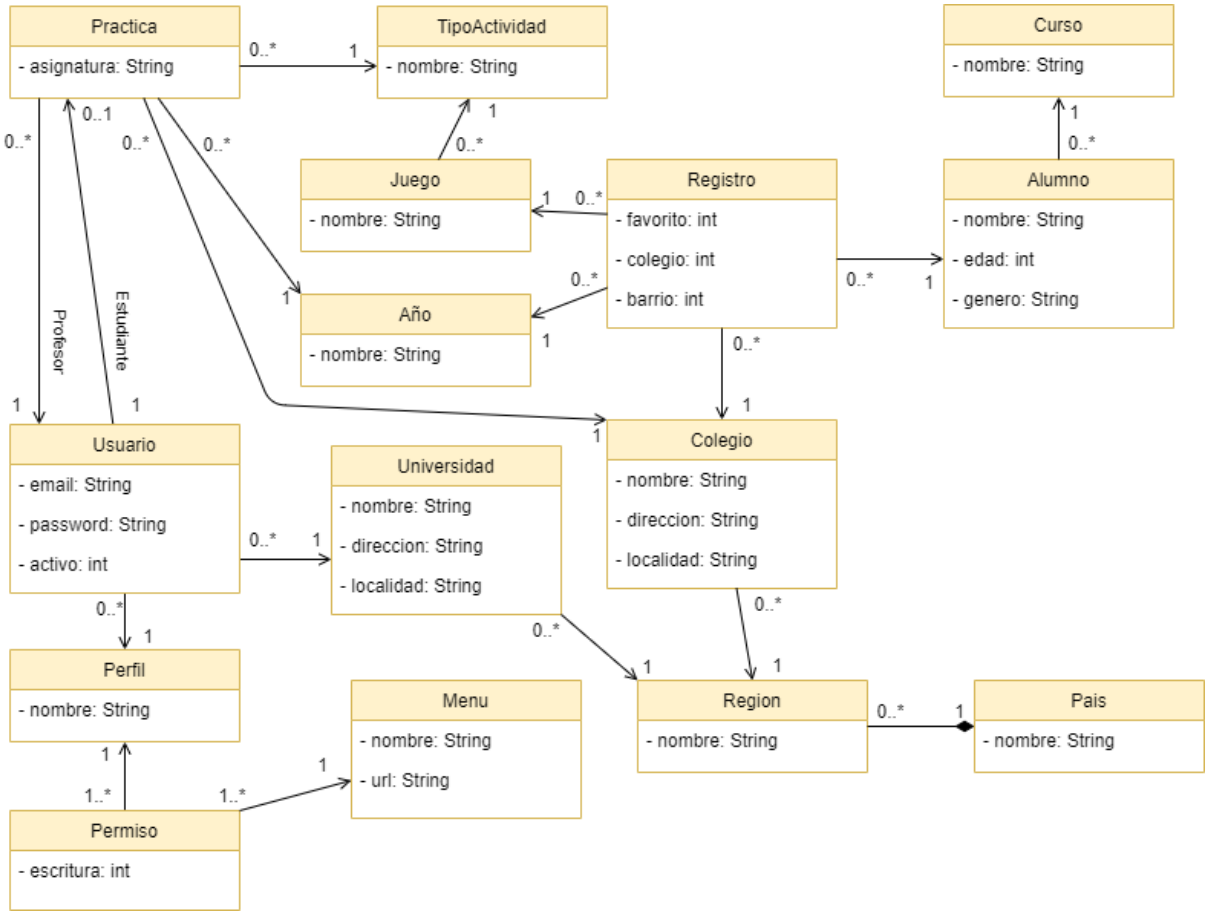


Figura 4.2: Diagrama de clases de la aplicación. Además de los atributos mostrados, todas las clases contienen tres atributos de tipo *Timestamp* (fechaAlta, fechaBaja, fechaModificacion), que no se han incluido para simplificar. También contienen una referencia a Usuario, que marca el autor de cada tupla.

que podremos navegar entre las secciones de la aplicación, a saber, Tablas, Perfil y Ayuda, además de poder salir de nuestra sesión en cualquier momento.

Dentro de cada tabla, podremos ver los datos más importantes de cada tupla, y si el usuario tiene permiso, podrá añadir nuevas tuplas o editar las ya existentes a través de formularios que se abrirán de forma emergente sobre la propia página. Se intentará que haya que recargar la página lo menos posible, pudiendo manipular los datos de una tabla sin tener que recargar constantemente la página, haciendo así la experiencia de uso más fluida.

5

Desarrollo

Tras haber visto las dos primeras fases del proyecto, el Análisis y el Diseño, nos centramos ahora en la tercera fase: el Desarrollo o Implementación. Se trata de traducir las especificaciones del diseño en lenguajes de programación para obtener el producto que buscábamos. Por tanto, en este capítulo vamos a ver qué lenguajes de programación, frameworks y tecnologías se han usado para la realización del proyecto, así como las soluciones aportadas para los principales características de la aplicación.

5.1. Entornos de desarrollo

Para la realización de este trabajo se han utilizado distintos entornos de desarrollo. Estos entornos, también conocidos por las siglas inglesas IDE (*Integrated Development Environment*), son programas que proporcionan herramientas integrales para facilitar el desarrollo del software a los programadores e ingenieros de software. Para la base de datos se ha utilizado **MySQL Workbench** [12], la herramienta para el diseño, visualización y gestión de bases de datos de Oracle para bases de datos hechas con MySQL. Para el desarrollo del back-end, que se ha hecho en Java, se ha utilizado **IntelliJ** [13], un entorno de desarrollo integrado para Java desarrollado por JetBrains. Aunque es un entorno diseñado especialmente para Java, también aporta asistencia de codificación para lenguajes de front-end como Javascript o HTML, así que se ha usado este mismo IDE para el front-end.

5.2. Lenguajes, frameworks y tecnologías

En esta sección vamos a ver qué tecnologías y *frameworks* se han usado para la realización de este proyecto. Recordemos que un framework, o entorno de trabajo en español, es un conjunto de conceptos y prácticas que sirven de base para tratar un problema de determinadas características y facilitar una tarea en el desarrollo del software.

5.2.1. Base de datos

La base de datos es uno de los pilares en los que se apoya un desarrollo software de este tipo. En este proyecto hemos optado por usar una base de datos relacional, y prácticamente todas las bases de datos de este tipo usan el lenguaje **SQL** (*Structured Query Language*) para obtener y mantener los datos.

MySQL

Aunque el lenguaje común en casi todas las bases de datos es el SQL, existen diversas herramientas de gestión basadas en este lenguaje. Para este proyecto se ha usado **MySQL**, que actualmente es la más popular y más usada para aplicaciones web [14].

5.2.2. Back-end

En un desarrollo web como el de este trabajo, el *back-end* es la parte que no está visible de cara al usuario, pero que se encarga de que todos los engranajes de la aplicación funcionen correctamente.

Java y Spring MVC

El lenguaje elegido para desarrollar esta aplicación ha sido **Java**. Se trata de un lenguaje de programación interpretado, orientado a objetos, basado en clases, cuyas principales características son la portabilidad, seguridad y robustez, lo cual ha producido que se mantenga durante veinte años como uno de los dos lenguajes de programación más populares, siempre en pugna con C, según el índice TIOBE [15].

El framework de Java con el que se ha trabajado es **Spring MVC**. Spring [16] es un framework para el desarrollo de aplicaciones construidas en Java basado en el principio de inversión de control y cuya principal característica es la inyección de dependencias. A su vez, Spring MVC es un framework bajo el paraguas de Spring, que facilita la creación de aplicaciones web que siguen el patrón de arquitectura Modelo-Vista-Controlador.

Maven

Si bien Java y Spring nos ofrecen el lenguaje y el framework necesarios para desarrollar nuestra aplicación, nos falta una herramienta para gestionarla. **Maven** [17] nos aporta este aspecto igual de importante en cualquier desarrollo. Se encargará de tareas tan importantes como la compilación, la ejecución de tests, la creación de ejecutables o ficheros listos para desplegar en el servidor.

Apache Tomcat

Por último, como servidor web usaremos **Apache Tomcat** [18], una implementación de los servlet de Java que permite correr aplicaciones web construidas en este lenguaje.

5.2.3. Front-end

Si el back-end era la parte no visible de la aplicación, cuando hablamos de front-end nos referimos a la parte visible, es decir, la interfaz de usuario de la aplicación. Para construir la interfaz de una aplicación web serán necesarias varias herramientas: un lenguaje de marcado para presentar los distintos elementos al usuario; un lenguaje de diseño para que esa presentación resulte más atractiva y visual; un lenguaje que permita hacer dinámica la interfaz, pudiendo interactuar con los elementos; y diversas herramientas para comunicarse con el back-end explicado en la subsección anterior.

HTML + Bootstrap (CSS)

El lenguaje de marcado **HTML** (*HyperText Markup Language*) es el estándar para la construcción de páginas web en la parte del cliente. También necesitamos **CSS** (*Cascading Style Sheets*), un lenguaje de diseño que permite dar estilos a los elementos HTML. Actualmente hay numerosas bibliotecas CSS que nos ahorran tiempo y esfuerzo a la hora de estilizar una web y hacerla atractiva de cara al usuario. Una de las más populares, y la que se ha utilizado en este trabajo, es **Bootstrap** [19]. También proporciona un sistema de cuadrículas para organizar el contenido web de forma que este sea *responsive*, es decir, que se adapte a los distintos tamaños de pantalla y a los cambios que esta pueda experimentar.

Vue.js (JavaScript)

Como hemos dicho, HTML es un lenguaje de marcado, es decir, no contiene ninguna lógica ni experimentará cambios ante interacciones del usuario. Para ello necesitaremos usar JavaScript, el lenguaje de programación interpretado que se usa en el cliente para hacer las webs dinámicas. Más concretamente, usaremos el framework **Vue.js** [20], que nos proporcionará herramientas para hacer esta programación más cómoda.

AJAX

Por último, necesitamos también una herramienta para comunicarnos con el servidor sin necesidad de recargar la web. Esta herrmienta será **AJAX** (*Asynchronous JavaScript And XML*), con la que podremos comunicarnos de manera asíncrona con el servidor y obtener respuestas para actualizar la presentación de los datos al usuario.

5.2.4. Spring Boot

Aunque con los elementos enumerados hasta ahora ya tendríamos lo necesario para levantar una aplicación web, integrar todos ellos conlleva unas tareas de configuración que pueden ser muy costosas, sobre todo en el *back-end*, llevando a errores y a pérdidas de tiempo que nos gustaría evitar. Por ello, para crear esta aplicación se ha usado **Spring Boot** [21], una herramienta que sigue el paradigma de *conveción sobre configuración* en programación, donde se busca minimizar la cantidad de decisiones que el programador tiene que realizar, simplificando el proceso de desarrollo y la mantenibilidad de cara al futuro.

Entre las facilidades que Spring Boot ofrece, encontramos la posibilidad de crear un proyecto base, eligiendo las características iniciales del mismo, como la versión de Java, si queremos que se trate de un proyecto de Maven o de Gradle, además de las dependencias que queremos añadir al proyecto. Spring Boot se encargará de auto configurar esas dependencias, añadiendo las librerías necesarias para su correcto funcionamiento. Las dependencias que hemos añadido para este proyecto son las siguientes:

- **Spring Boot DevTools**: provee herramientas para facilitar la expriencia de desarrollo, como reinicio de la aplicación si cambiamos algún fichero.
- **Lombok**: evita que tengamos que escribir código repetitivo, pudiendo sustituir getters, setters o constructores con sencillas anotaciones en las clases. Por ejemplo, si anotamos una clase con @Getter y @Setter, Lombok construirá automáticamente los métodos públicos para obtener o editar cualquiera de los atributos de la clase, pero no se mostrarán en el entorno de desarrollo, haciendo más pulcra la visualización de nuestras clases. Si anotamos una clase con @AllArgsConstructor y @NoArgsConstructor, Lombok creará los constructores por defecto para esa clase, tanto el constructor sin argumentos como el que tiene todos. También hay anotaciones para métodos toString, hashCode, equals, etc.
- **Spring Web**: nos proporciona herramientas para construir aplicaciones web, especialmente útiles cuando seguimos el patrón MVC . Por ejemplo, tendremos anotaciones para los controladores como @Controller, @GetMapping, @PostMapping, etc.
- **Thymeleaf**: se trata de un motor de plantillas para HTML que nos va a facilitar tanto la reutilización de código repetitivo en las distintas vistas del proyecto, como la presentación de datos enviados desde el servidor.
- **Spring Security**: es un framework de autenticación y control de acceso para aplicaciones construidas con Spring. Viene con una configuración por defecto que podemos modificar de manera sencilla para adaptarla a nuestras necesidades. Nos será de gran utilidad para gestionar los accesos a la aplicación y los permisos de cada tipo de usuario.
- **MySQL Driver**: permitirá la conexión con el gestor de bases de datos elegido, MySQL.

- **Spring Data JPA**: la **Java Persistance API (JPA)** es la API de persistencia de datos de Java. Por otro lado, **Hibernate** es una herramienta de mapeo entre modelos de bases de datos relacionales y modelos de datos de una aplicación Java. Spring Data JPA agrupa estas tecnologías bajo el paraguas de Spring Data (un proyecto más abstracto, basado en tecnología Spring, que sienta las bases para el acceso a datos en este tipo de aplicaciones), proporcionando herramientas para crear repositorios de datos en conexión con nuestra base de datos de manera sencilla.
- **Java Mail Sender**: este framework nos permitirá enviar emails desde nuestra aplicación de manera sencilla, lo cual nos será útil en el proceso de registro de usuarios.

5.3. Implementación de la arquitectura

En esta subsección vamos a ver cómo se ha llevado a cabo la implementación del patrón de arquitectura elegido para el desarrollo que, como ya hemos dicho, se trata del patrón **Modelo-Vista-Controlador**. En el proyecto veremos cuatro carpetas principales de clases Java: *model*, *repository*, *controller* y *service*, cuyas funciones vamos a explicar a continuación.

5.3.1. El modelo

En la carpeta *model* encontraremos el modelo de datos. Tenemos una clase Java para cada entidad de la base de datos. Hacemos uso de las distintas anotaciones que nos ofrece la API de persistencia de Java (**JPA**) para hacer la traducción de esas entidades en clases Java, manteniendo las propiedades y relaciones de cada entidad.

Veamos un ejemplo de uso de estas anotaciones. Tomemos una relación Many To One (muchos a uno) de nuestro modelo de datos. Por ejemplo, como se puede ver en el diagrama de clases de la figura 4.2, cada *usuario* pertenece a una *universidad*, mientras que a cada *universidad* pueden pertenecer diversos usuarios. Esta relación se mapea con una *clave foránea* en el *usuario*, pero cuando transformamos esa entidad en una clase Java no nos interesa tener solo el *id* de la clave foránea, sino que queremos tener la universidad como tal para poder acceder a sus atributos directamente. Esto lo conseguimos creando un atributo en la clase de Java Usuario de tipo Universidad, que irá marcado con las anotaciones @ManyToOne y @JoinColumn, como podemos ver en la figura 5.1. De esa forma, Hibernate sabrá el tipo de relación existente y la clave foránea, y obtendrá el objeto Universidad que corresponda. Además, se especifica el tipo de *fetching*, es decir, cómo queremos que se haga la búsqueda de ese objeto.

Por defecto, JPA define las relaciones Many To One con el modo de *fetching EAGER*. Este modo carga la entidad hija cada vez que se carga la entidad padre. Esto puede provocar consultas SQL

```
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fkuniversidad", nullable = false)
private Universidad universidad;
```

Figura 5.1: Ejemplo de anotaciones JPA. En este caso, @JoinColumn indica que se trata de una referencia a otra entidad, y @ManyToOne indica el tipo de relación y el tipo de *fetching*.

innecesarias y más peso en los objetos que manejamos. Por tanto, se ha usado siempre que ha sido posible el modo de *fetching LAZY*, que solo carga el objeto hijo cuando este es llamado explícitamente. Así, si obtenemos un *usuario* pero solo manipulamos su email y contraseña, la *universidad* no se cargará.

Con las clases del modelo ya tenemos la definición del mismo en Java, pero nos faltan los métodos para traer los distintos objetos desde la base de datos. Para ello, en la carpeta *repository* encontramos los repositorios de datos. Encontramos un repositorio para cada clase del modelo, y todos ellos extienden la interfaz ***CrudRepository***. Esta interfaz que nos proporciona Spring contiene métodos básicos para acceder y modificar datos (*CRUD* es el acrónimo de *Create, Read, Update, Delete*, que son las acciones elementales que podemos realizar sobre la base de datos: Crear, Leer, Actualizar y Borrar), y nos permite crear nuestros propios métodos. Solo tenemos que encadenar una serie de directivas para formar el nombre del método, y Spring se encargará del resto. Por ejemplo, si queremos recuperar todos los *colegios* creados por cierto usuario y que no hayan sido dados de baja, solo tenemos que declarar el método findAllByUsuarioIdAndFechaBajaIsNull(Integer id), y Spring se encarga de crear la consulta.

5.3.2. El controlador

Los controladores reciben las peticiones del cliente y, tras ejecutar cierta lógica de negocio, seleccionan una vista que se puebla con los datos requeridos. Para mantener el código lo más limpio posible, se ha llevado a cabo una **arquitectura en capas**. Se ha creado una capa intermedia entre el controlador y los repositorios de datos, donde se encapsula toda la lógica de negocio en clases llamadas Servicios. Así, cuando un controlador recibe una petición, este llama a los servicios necesarios, que ejecutan la lógica de negocio y llaman a los repositorios, que son la conexión con la base de datos. Estos ejecutan las acciones indicadas o devuelven los datos pedidos a los servicios, que a su vez los devuelven al controlador. Por último, el controlador sirve la vista que haga falta con los datos obtenidos. Agrupamos todos los controladores en la carpeta *controller*, y todos los servicios en *service*.

5.3.3. La vista

Tendremos un documento html para cada vista, que ubicadas en la carpeta *templates* serán buscadas automáticamente por Spring cuando un cotrolador las solicite. Las vistas presentes en la aplicación serán el login, la página de inicio, la ayuda, el perfil, etc., pero las más importantes sin ninguna duda serán las de las distintas tablas que el usuario podrá consultar, y que se explicarán en más detalle en la siguiente sección.

5.4. Tablas dinámicas con AJAX y Vue

Cuando el usuario navega por los datos de una tabla haciendo consultas, manipulaciones o creando nuevos datos, queremos que esa navegación sea fluida, que no tenga que estar cambiando de página constantemente para crear, editar elementos o ver detalles. Para ello se ha optado por una implementación con AJAX y Vue. Con el primero nos comunicaremos de manera asíncrona con el servidor y recibiremos respuestas, que usaremos para actualizar dinámicamente las tablas gracias al framework de JavaScript de Vue.

Idealmente, toda la web sería una aplicación Vue, y las distintas páginas y tablas serían componentes Vue que se irían mostrando y actualizando según las interacciones del usuario. Este concepto es el de las SPA, *Single Page Application*, donde la web solo carga una vez y luego se va actualizando dinámicamente. Sin embargo, cuando hay demasiadas variables a tener en cuenta para mantener una página actualizada de manera asíncrona y dinámica, es bastante complicado no cometer ningún fallo en esas actualizaciones, teniendo el riesgo de mostrar al usuario información errónea o poco precisa, sobre todo cuando es una aplicación que puede ser usada por varios usuarios de forma simultánea.

Por ello, y dado que la curva de aprendizaje de Vue o cualquier otro framework de JavaScript para llevar a cabo aplicaciones de ese tipo se sale del alcance de este trabajo, se ha optado por introducirlo a una escala menor. Así, dentro de cada vista, la tabla de datos será nuestro componente principal Vue.

En el componente principal se cargan inicialmente los datos que vienen del controlador. Para no cargar demasiado al cliente con exceso de datos, se han creado unas clases para mandar a cada vista la información justa y necesaria. Es decir, cuando el controlador sirve la vista correspondiente a la tabla de usuarios, no envía un array con los usuarios tal cual salen del modelo, sino que crea unos objetos auxiliares que solo llevarán la información que se mostrará en las tablas.

Dentro del componente principal podemos realizar búsquedas “en directo” por cualquier campo que se muestre y navegar entre las distintas páginas (que se ajustan dinámicamente según la búsqueda que haya en el momento, como podemos ver en la figura 5.2). Para estas características se ha hecho uso de una biblioteca de filtros para Vue [22], unos filtros que aunque formaban parte del framework

original en versiones anteriores, desaparecieron en la versión Vue2.

Veremos también un botón para añadir una nueva tupla de datos. Si clicamos se abrirá una ventana modal con un formulario que tendrá unos campos u otros dependiendo de en qué tabla nos encontremos. Este modal es un componente Vue hijo del componente principal, y es el mismo que aparecerá si clicamos en el botón de editar una fila, pero en este caso aparecerá con los datos rellenos del elemento que vayamos a editar. Podemos ver un ejemplo de estas ventanas modales en la figura 5.3.

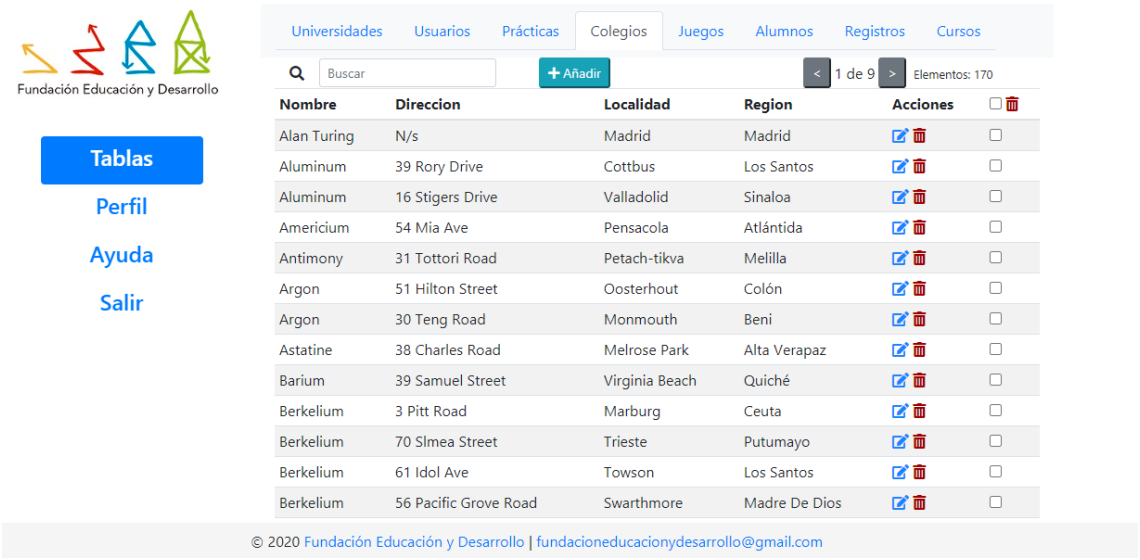
Al lado del botón de editar una fila encontramos un botón de borrado. Si clicamos en este botón se abrirá otro modal con un mensaje informativo del elemento que vamos a borrar, y botones con las opciones de confirmar el borrado o cancelar. También tenemos la opción de realizar un borrado de varios elementos a la vez. Para ello, podemos seleccionar tantas filas como deseemos con los checkbox de la derecha, y pulsar en el botón de borrar al inicio de la última columna. Si hacemos esto, se abrirá el mismo modal anterior, pero informándonos de cuántos elementos hemos seleccionado y dándonos de nuevo la opción de confirmar el borrado o de cancelarlo.

Recordemos que, cumpliendo con lo establecido en el requisito RNF-2.1, todos los borrados serán borrados lógicos. Es decir, no se eliminarán físicamente los elementos de la base de datos, sino que se marcará una fecha de baja. En todas las tablas de la aplicación solo se cargarán aquellos elementos que no tengan fecha de baja.

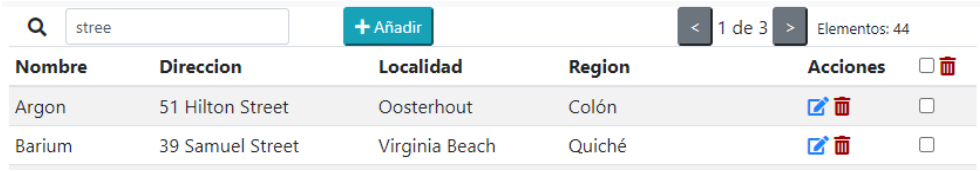
Una vez confirmada la creación de un nuevo elemento, la edición de uno ya existente o el borrado de uno o más elementos, se lanzará un método **AJAX** que solicitará al servidor ejecutar esa acción mediante peticiones PUT, POST y DELETE. El servidor responderá con un objeto de la clase ServiceResponse, una clase que contiene toda la información necesaria para actualizar la vista desde el cliente: el objeto creado o editado, el mensaje de éxito o error, o el tipo de acción que se ha llevado a cabo. En base a esta respuesta, que recibirá el componente principal de Vue, se actualizará la tabla de la manera oportuna: añadiendo una fila, modificando una ya existente, o eliminando las que se hayan borrado. También aparecerá en la parte superior de la tabla un mensaje informando del éxito o fracaso de la operación.

5.5. Seguridad

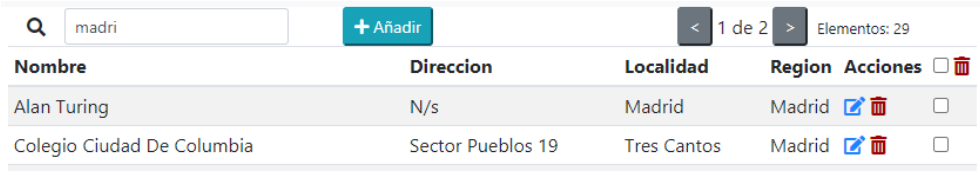
En cualquier aplicación que maneje datos y cuentas, la seguridad es un aspecto clave para la confianza del cliente y el usuario. Por ello, es importante definir unos mecanismos que garanticen la confidencialidad en todos los aspectos de la aplicación.



(a) Vista de la tabla sin realizar ninguna búsqueda. Vemos que hay 9 páginas de datos con un total de 170 colegios.



(b) Introducimos en la caja de búsqueda la cadena *stre* e inmediatamente vemos cómo con cada caracter se va actualizando la cantidad de elementos mostrados. Vemos que el número de páginas se ha ajustado de manera automática a 3, y el número de elementos a 44. En este caso, al menos las dos primeras filas aparecen porque contienen la cadena buscada en el campo Dirección.



(c) Otro ejemplo de búsqueda. Aquí hemos introducido la cadena *madri* ya que queremos ver los colegios de la región de Madrid. Observamos que hay 2 páginas de colegios con un total de 29 elementos que coinciden con la búsqueda.

Figura 5.2: Vista general de la aplicación y ejemplo de tabla dinámica, en este caso de colegios. Podemos observar los diferentes elementos que la componen: los datos, la caja de búsqueda, la paginación, así como las opciones de añadir, editar y borrar. En las subfiguras 5.2(b) y 5.2(c) podemos ver qué ocurre al realizar búsquedas.

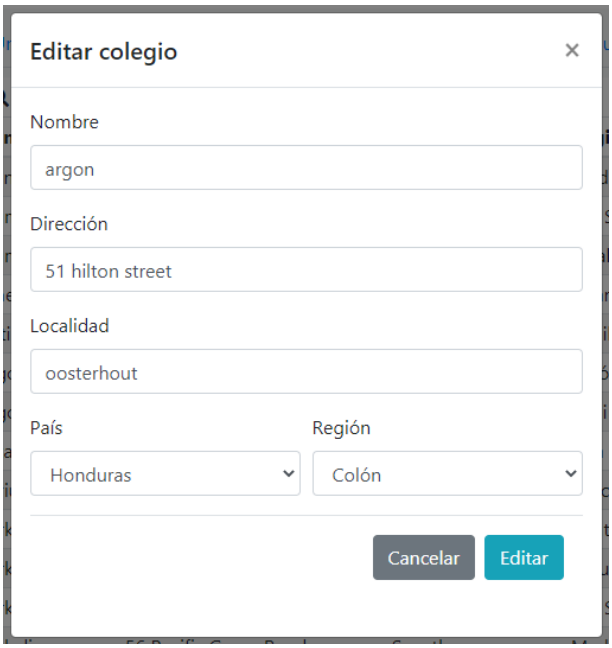


Figura 5.3: Ejemplo de ventana modal. En este caso, se trata de la ventana para editar elementos de la tabla Colegios. Según la tabla de que se trate, podemos encontrar inputs de distintos tipos: texto, número, select, checkbox, etc.

5.5.1. Registro de usuarios

La aplicación tiene la particularidad de que no es el usuario quien se registra, sino que otro usuario con ciertos permisos tiene que registrarlo. Por ejemplo, un administrador puede registrar a nuevos profesores, y un profesor a nuevos estudiantes. Por tanto, en el momento del registro no es el usuario final quien elige la contraseña, sino que se genera una contraseña de 10 caracteres alfanuméricos y con caracteres especiales de forma automática y aleatoria, para lo cual se ha hecho uso de una biblioteca de Java que refuerza el cumplimiento de normativas de contraseñas llamada Passay [23]. En ese momento se envía un email automático al nuevo usuario, informándole de la creación de la cuenta y de su contraseña, y recomendándole cambiarla en el área de su perfil de la aplicación. Tras enviar el email, la contraseña se cifra y almacena como se explicará más adelante.

5.5.2. Cifrado de contraseñas y base de datos

Uno de los puntos más importantes en la seguridad de una aplicación es la manera de manipular y almacenar las contraseñas. Si alguien consiguiera acceder a la base de datos y encontrara las contraseñas tal y como son realmente, estaríamos dando acceso a la aplicación con cualquier cuenta. Por ello, no nos basta con implantar un sistema de registro y login para restringir el acceso solo a las personas que tengan una cuenta. También hay que preocuparse de cómo se almacenan y comprueban las contraseñas en la base de datos.

La manera usual de proteger las contraseñas es almacenando, en vez del texto plano, el resultado de una función hash cuyo argumento es la contraseña. Estas funciones toman la contraseña y producen una salida única, a partir de la cual es imposible obtener la contraseña original. Ese resultado es lo que se guardará en la base de datos. Cuando el usuario introduzca su contraseña para entrar en la aplicación, se calculará el resultado de la función hash sobre lo que haya introducido, y si coincide con lo almacenado en la base de datos, se permitirá el acceso a la aplicación.

Para nuestra aplicación se ha decidido usar la función hash de codificación **BCrypt**, diseñada en 1999 por Niels Provos y David Mazières [24] y basada en *Blowfish*, otro algoritmo de codificación. Para ello se ha usado la implementación de esta función que tiene Spring Security, *BCryptPasswordEncoder*, más segura que otros codificadores basados en otros algoritmos hash como Md5 o Sha, que ya han sido declarados obsoletos. Se cumple así el requisito de seguridad RNF-3.2.

Otro aspecto de la seguridad relacionado con la base de datos es que la aplicación está protegida contra ataques por *inyecciones SQL*, ya que en ningún momento se usan queries nativas en la lógica de negocio. Al usar siempre métodos construidos con los repositorios de JPA, se escapará cualquier intento de inyección.

5.5.3. Permisos

Como se especifica en el requisito RF-2, en la aplicación hay diferentes roles asignados a los usuarios. Según qué rol tenga un usuario, podrá tener acceso a unas u otras funcionalidades. Para satisfacer de esa manera el requisito RNF-3.3, se ha utilizado el framework de Spring Security para limitar el acceso de cada rol solo a los menús especificados desde base de datos. Además, en el *front-end* se comprueba de forma dinámica el rol del usuario actual para mostrar unos u otros contenidos según los permisos que ese rol tenga. Así se consigue una doble seguridad: se evita que el usuario vea enlaces a páginas para las que no tiene permiso, y se prohíbe el acceso a esas páginas a través de url.

5.6. Textos e internacionalización con Thymeleaf

Construir una web en varios idiomas puede ser muy cansado si se duplican la cantidad de htmls cada vez que añadimos un idioma nuevo. Además, si lo hacemos de esa manera y queremos realizar un cambio en una vista, tendríamos que cambiar las vistas correspondientes a cada uno de los idiomas. Afortunadamente, usando Spring y Thymeleaf podemos crear webs que seleccionan los mensajes de forma dinámica para encajar con el idioma seleccionado por el usuario.

Para ello, contamos con un fichero .properties para cada idioma que queramos utilizar. Este fichero contendrá todos los mensajes que aparecen en la web, cada uno con una etiqueta única. En los html,

en vez de mostrar texto introducido a mano, llamaremos de forma dinámica a estas etiquetas usando Thymeleaf, que seleccionará el mensaje del fichero .properties del idioma en el que se esté mostrando la web. El idioma por defecto será el español. Esto hará que si Thymeleaf no encuentra una etiqueta traducida, mostrará el mensaje de esa etiqueta en español. Como último recurso, si no encuentra ninguna mensaje con esa etiqueta en el fichero, usará el texto plano introducido en el html.

Este mecanismo hace cumplir los requisitos RNF-4.1 y RNF-6.1 pues, además de permitir añadir idiomas fácilmente, también centraliza todos los mensajes de la web en un único fichero y permite editarlos de manera sencilla sin tener que acceder a los html.

6

Pruebas y resultados

Durante y tras el desarrollo de un proyecto software, es necesario realizar distintas pruebas con el objetivo de encontrar errores y solucionarlos lo antes posible. En este capítulo vamos a exponer las pruebas que se han realizado y los resultados de las mismas.

6.1. Base de datos

La base de datos de esta aplicación tiene bastantes tablas, todas ellas relacionadas entre sí. Por tanto, la definición de la misma es de vital importancia para el correcto funcionamiento del sistema. También es importante tener una muestra de datos para poder comprobar que todo funciona como debe hacerlo.

Durante el desarrollo, cada vez que se ha completado un módulo (usuarios, colegios, universidades, juegos, etc.), se han hecho pruebas correspondientes a las operaciones CRUD en base de datos. En estas pruebas se han encontrado algunos errores de definición, como erratas en los tipos de los atributos o en las relaciones con claves foráneas, que se han resuelto sin mayor problema.

Para las pruebas que se van a explicar más adelante, se ha poblado la base de datos usando una herramienta de generación de datos aleatorios, introduciendo del orden de 4.000 alumnos, 200 colegios, 1.000 juegos, 500 prácticas, 7.000 registros de datos, 600 universidades y 100 usuarios. Para los países y regiones se han tomado los códigos ISO 3166-1 e ISO 3166-2 [25], lo que supone unos 200 países y unas 4.000 regiones.

6.2. Pruebas unitarias

El objetivo de las pruebas unitarias es comprobar el correcto funcionamiento de forma individual de cada módulo, componente o servicio. Para cada tabla se ha comprobado el correcto funcionamiento de las operaciones de creación, edición y eliminación de datos. También se ha comprobado el correcto funcionamiento del mecanismo de cambio de contraseña, con sus posibles errores.

6.3. Pruebas de integración

Las pruebas de integración tienen como objetivo comprobar el funcionamiento del sistema en su conjunto, viendo que todos los módulos o componentes interactúan de manera correcta y sin fallos.

Se ha comprobado que solo los usuarios registrados tienen acceso a las páginas de la aplicación más allá del login, y que un usuario con determinado rol solo tiene acceso a las áreas que se le permiten. Se verifica que ante cualquier error significativo, el usuario es dirigido a una página informativa sobre la misma interfaz de la aplicación, y nunca a una página de error genérica de Spring Boot o Tomcat. Se comprueba también el funcionamiento correcto de las tablas, ya no solo las operaciones de *back-end*, sino también que se actualizan en el *front-end* de manera correcta ante cualquier interacción. Se comprueba que el sistema de registro funciona correctamente, que se envía el email informativo al usuario creado, que se cifra la contraseña y que el sistema de cambio de contraseña funciona bien.

6.4. Pruebas con usuarios

La idea cuando empezamos a relizar este proyecto era que la aplicación, o al menos una primera versión usable, sería probada durante el segundo semestre por estudiantes en prácticas de la Universidad Autónoma de Madrid. Sin embargo, debido a la situación derivada por la pandemia de Covid-19, y la consecuente cancelación de todas las prácticas, estas pruebas con usuarios no han sido posibles. En cualquier caso, la aplicación ha sido probada por personas de la FED , que aunque no serían usuarios de la aplicación del mismo modo que un estudiante, sí administradores de la misma. Tras hacer estas pruebas se han encontrado errores menores en alguna funcionalidad, como la falta de alguna validación en formularios o de algún mensaje informativo, que se han podido solucionar sin mayor problema.

6.5. Resultados

Los resultados de las pruebas son satisfactorios, y concluímos que nos encontramos ante un producto funcional que cumple con los requisitos que se impusieron al comienzo del proyecto. Hemos obtenido una aplicación lista para pasar a una fase de pruebas más amplia, donde se comenzará a utilizar con usuarios reales en el curso 2020/2021 si la situación en la que nos encontremos lo permite.

7

Conclusiones y trabajo futuro

Tras pasar por las distintas fases del desarrollo software, hemos obtenido un producto funcional que cumple con los requisitos que se propusieron al principio del proyecto. En este último capítulo vamos a hacer un repaso del trabajo realizado y vamos a plantear posibles líneas de trabajo para seguir mejorando la herramienta en un futuro.

7.1. Conclusiones

Con este proyecto hemos querido dar una solución a un problema que se planteaba en la Fundación Educación y Desarrollo. Gracias a la herramienta creada se podrá centralizar la recogida y gestión de los datos obtenidos por estudiantes en prácticas sobre preferencias de juegos en niños y niñas.

Se comenzó el proyecto analizando el problema, creando unos requisitos iniciales y diseñando una base de datos que permitiera poner en funcionamiento la aplicación web. En el desarrollo se han utilizado tecnologías muy diversas para solucionar los distintos aspectos que entran en juego cuando construimos un sistema de este tipo, y con las pruebas nos hemos asegurado de que el sistema construido cumple con los requisitos que habíamos especificado.

En cuanto a los conocimientos adquiridos a nivel personal, he profundizado en todo lo que conlleva realizar un proyecto software de principio a fin. La tremenda importancia que tienen (y que a veces no se les da) las fases de análisis y diseño para evitar problemas posteriores es uno de esos aprendizajes. También la importancia de tener un equipo al lado para aumentar los ángulos con los que se enfoca un problema, aunque este punto se haya aprendido por echarlo de menos y no por su presencia.

7.2. Trabajo futuro

La base de datos y la aplicación web desarrollada en este trabajo puede considerarse un punto de partida para un proyecto que puede ser mucho más ambicioso. Este trabajo sienta las bases para empezar a perfeccionar un sistema que puede ser de mucha utilidad.

Un área fundamental que se plantea al iniciar el proyecto y que se deja como pendiente para un futuro desarrollo es dotar al rol de investigador de herramientas para poder obtener análisis estadísticos a partir de los datos que puede consultar a través de la aplicación. Estas estadísticas podrían visualizarse directamente en la aplicación, o a través de informes generados en la misma, y podrían estar basados en género, edad o localización geográfica de los niños y las niñas.

Durante la realización del trabajo ha surgido una cuestión a la que, de momento, no se ha dado solución. En la base de datos existen diversos juegos y actividades, cada uno con un identificador único y un nombre. Cuando un estudiante va a introducir un registro puede buscar el juego por nombre, y si no lo encuentra, tiene la posibilidad de crear un juego nuevo y luego seleccionarlo en la creación del registro de datos. Sin embargo, entra en juego un factor social que puede descuadrar la generación de estadísticas: hay juegos que no se llaman igual según en qué país, o incluso región, nos encontremos. Surge aquí la posibilidad de hacer lo que podría ser un “diccionario de juegos”, creando una especie de relación de equivalencia entre juegos existentes en la base de datos que, si bien tienen distinto nombre, hacen referencia a la misma actividad, y serían tomados en cuenta de manera conjunta a la hora de generar estadísticas.

Una posible mejora de la interfaz de usuario sería la de implementar la funcionalidad de añadir elementos “en paralelo”. Es decir, que cuando un estudiante vaya a introducir los datos de un formulario, no tenga que relizar una acción de añadir por cada juego mencionado por el niño o niña, sino que en la misma ventana modal se puedan añadir simultáneamente varios juegos junto con sus marcas correspondientes de si es un juego favorito, si se juega en el barrio o en el colegio.

Otra línea interesante de trabajo futuro podría ser la de realizar un nuevo módulo para importar los datos recogidos hasta ahora en las hojas de cálculo, y así poder hacer los análisis de datos de manera retroactiva. Se podría hacer un script que transformara esos ficheros en datos útiles que se insertaran en la base de datos. Para ello habría que hacer unas equivalencias entre los códigos de colegios utilizados en los ficheros y los ids de los colegios en la base de datos. También habría que ver cómo se gestiona la existencia de juegos con nombres parecidos. En la aplicación, al poder buscar el nombre del juego, es raro que un estudiante añada el juego “pilla pilla” si ya existe el juego “pillapilla”. En los ficheros, sin embargo, al ser individuales, esto no podía hacerse, y hay casos en los que un mismo juego tiene diversos nombres muy parecidos. Este nuevo módulo podría integrarse con el “diccionario de juegos” mencionado anteriormente para solucionar este problema.

Por último, queda pendiente realizar el periodo de pruebas con usuarios finales que no se han podido llevar a cabo este año, como se ha comentado, por la situación derivada de la pandemia.

Bibliografía

[1] *stackoverflow*. https://stackoverflow.com/.

[2] *Fundación Educación y Desarrollo*. http://www.fundacioneducacionydesarrollo.org/.

[3] J. L. Linaza, "El juego es un derecho y una necesidad de la infancia," *Bordón. Revista de pedagogía*, 2013.

[4] Twitch, *IGDB API*. https://api.igdb.com/.

[5] *Twitch*. https://www.twitch.tv/p/es-es/about/.

[6] *Playworks*. https://www.playworks.org/.

[7] *Librería de juegos de Playworks*. https://www.playworks.org/game-library/.

[8] B. Sutton-Smith and B. G. Rosenberg, "Sixty years of historical change in the game preferences of american children," *The Journal of American Folklore*, vol. 74, no. 291, pp. 17–46, 1961.

[9] M. B. Kinzie and D. R. Joseph, "Gender differences in game activity preferences of middle school children: implications for educational game design," *Educational Technology Research and Development*, vol. 56, no. 5-6, pp. 643–663, 2008.

[10] Z. Tatli, "Traditional and digital game preferences of children: A chaid analysis on middle school students.," *Contemporary Educational Technology*, vol. 9, no. 1, pp. 90–110, 2018.

[11] G. E. Krasner, S. T. Pope, *et al.*, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.

[12] *MySQL Workbench*. https://www.mysql.com/products/workbench/.

[13] *IntelliJ IDEA*. https://www.jetbrains.com/es-es/idea/.

[14] *The Most Popular Databases 2019*. https://www.explore-group.com/blog/the-most-popular-databases-2019/bp46/.

[15] *TIOBE Index for July 2020*. https://www.tiobe.com/tiobe-index/.

[16] *Spring Framework*. https://spring.io/.

[17] *Apache Maven*. https://maven.apache.org/.

[18] *Apache Tomcat*. http://tomcat.apache.org/.

[19] *Bootstrap*. https://getbootstrap.com/.

[20] *Vue.js, The Progressive JavaScript Framework*. https://vuejs.org/.

[21] *Spring Boot*. https://spring.io/projects/spring-boot.

[22] freearhey, *vue2-filters*. https://github.com/freearhey/vue2-filters.

[23] *Passay: password policy for Java*. http://www.passay.org/.

[24] N. Provos and D. Mazieres, "A future-adaptable password scheme.," in *USENIX Annual Technical Conference, FREENIX Track*, pp. 81–91, 1999.

[25] *ISO 3166*. https://www.iso.org/glossary-for-iso-3166.html.

Acrónimos

AJAX Asynchronous JavaScript And XML.

API Application Programming Interface.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

FED Fundación Educación y Desarrollo.

HTML HyperText Markup Language.

IDE Integrated Development Environment.

JPA Java Persistance API.

MVC Modelo-Vista-Controlador.

SPA Single Page Application.

SQL Structured Query Language.

TFG Trabajo de Fin de Grado.

UAM Universidad Autónoma de Madrid.